MEMORIZATION CAPACITY OF NEURAL NETWORKS WITH CONDITIONAL COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Many empirical studies have demonstrated the performance benefits of conditional computation in neural networks, including reduced inference time and power consumption. We study the fundamental limits of neural conditional computation from the perspective of memorization capacity. For Rectified Linear Unit (ReLU) networks without conditional computation, it is known that memorizing a collection of n input-output relationships can be accomplished via a neural network with $O(\sqrt{n})$ neurons. Calculating the output of this neural network can be accomplished using $O(\sqrt{n})$ elementary arithmetic operations of additions, multiplications and comparisons for each input. Using a conditional ReLU network, we show that the same task can be accomplished using only $O(\log n)$ operations per input. This represents an almost exponential improvement as compared to networks without conditional computation. We also show that the $O(\log n)$ rate is the best possible.

1 INTRODUCTION

1.1 CONDITIONAL COMPUTATION

Conditional computation refers to utilizing only certain parts of a neural network, in an input-adaptive fashion (Davis & Arel, 2013; Bengio et al., 2013; Eigen et al., 2013). This can be done through gating mechanisms combined with a tree-structured network, as in the case of "conditional networks" (Ioannou et al., 2016) or neural trees and forests (Tanno et al., 2019; Yang et al., 2018; Kontschieder et al., 2015). Specifically, depending on the inputs or some features extracted from the inputs, a gate can choose its output sub-neural networks that will further process the gate's input features. Another family of conditional computation methods are the so-called early-exit architectures (Teerapittayanon et al., 2016; Kaya et al., 2019; Gormez & Koyuncu, 2021). In this case, one typically places classifiers at intermediate layers of a large network. This makes it possible to exit at a certain layer to reach a final verdict on classification, if the corresponding classifier is confident enough of its decision.

Several other sub-techniques of conditional computation exist and have been well-studied, including layer skipping (Graves, 2016), channel skipping in convolutional neural networks (Gao et al., 2019), or reinforcement learning methods for input-dependent dropout policies (Bengio et al., 2015). Although there are many diverse methods (Han et al., 2021), the general intuitions as to why conditional computation can improve the performance of neural networks remain the same: First, the computation units are chosen in an adaptive manner to process the features that are particular to the given input pattern. For example, a cat image is ideally processed by only "neurons that are specialized to cats." Second, one allocates just enough computation units to a given input, avoiding a waste of resources. The end result is various benefits relative to a network without conditional computation, including reduced computation time, and reduced power/energy consumption (Kim & Seo, 2020). Moreover, these benefits arrive with minimal loss, or in some cases, without any loss in learning performance.

Numerous empirical studies have demonstrated the benefits of conditional computation in many different settings. Understanding the fundamental limits of conditional computation in neural networks is thus crucial, but has not been well-investigated in the literature. There is a wide body of work on a theoretical analysis of decision tree learning (Maimon & Rokach, 2014), which can be considered as an instance of conditional computation. These results are, however, not directly applicable to neural networks. In Cho & Bengio (2014), a feature vector is multiplied by different weight matrices, depending on the significant bits of the feature vector, resulting in an exponential increase in the

number of free parameters of the network (referred to as the capacity of the network in Cho & Bengio (2014)). On the other hand, the potential benefits of this scheme have not been formally analyzed.

1.2 MEMORIZATION CAPACITY

In this work, we consider the problem of neural conditional computation from the perspective of memorization capacity. Here, the capacity refers to the maximum number of input-output pairs of reasonably-general position that a neural network of a given size can learn. It is typically expressed as the minimum number of neurons or weights required for a given dataset of size, say n.

Early work on memorization capacity of neural networks include Baum (1988); Mitchison & Durbin (1989); Sontag (1990). In particular, Baum (1988) shows that, for thresholds networks, $O(n)$ neurons and weights are sufficient for memorization. This sufficiency result is later improved to $O(\sqrt{n})$ neurons and $O(n)$ weights by Vershynin (2020); Rajput et al. (2021). There are also several studies on other activation functions, especially the Rectified Linear Unit (ReLU), given its practicality and wide utilization in deep learning applications. Initial works Zhang et al. (2017); Hardt & Ma (2016) show that $O(n)$ neurons and weights are sufficient for memorization in the case of ReLU networks. This is improved to $O(n^{2/3})$ in Park et al. (2021), and finally, up to logarithmic factors, to $O(\sqrt{n})$ weights and neurons in Vardi et al. (2022). For the sigmoid activation function, it is known that $O(\sqrt{n})$ neurons and $O(n)$ weights (Huang, 2003), or $O(n^{2/3})$ weights and neurons (Park et al., 2021) are sufficient.

The aforementioned achievability results have also been proven to be tight in certain cases. A very useful too in this context is the Vapnik-Chervonenkis (VC) dimension (Vapnik & Chervonenkis, 2015). In fact, applying the VC dimension theory to neural networks (Anthony et al., 1999), it can be shown that a necessary condition for memorization is that the number of neurons and weights should be both polynomial in the size of the dataset. Specifically, $O(\sqrt{n})$ weights and $O(n^{1/4})$ neurons are optimal for ReLU networks, up to logarithmic factors. We will justify this statement later on for completeness.

1.3 SCOPE, MAIN RESULTS, AND ORGANIZATION

We analyze the memorization capacity of neural networks with conditional computation. We describe our neural network and the associated computational complexity models in Section 2. We describe a general method to synthesize conditional networks from unconditional networks in Section 3. We provide our main achievability and converse results for memorization capacity in Sections 4 and 5, respectively. We draw our main conclusions in Section 6. Some of the technical proofs are provided in the supplemental material.

We note that this paper is specifically on analyzing the theoretical limits on neural conditional computation. In particular, we show that n input-output relationships can be memorized using a conditional network that needs only $O(\log n)$ operations per input or inference step. The best unconditional architecture requires $O(\sqrt{n})$ operations for the same task. This suggests that conditional models can offer significant time/energy savings as compared to unconditional architectures. In general, understanding the memorization capacity of neural networks is a well-studied problem of fundamental importance and is related to the expressive power of neural networks. A related but separate problem is generalization, i.e. how to design conditional networks that can not only recall the memory patterns with reasonable accuracy but also generalize to unseen examples. The "double-descent" phenomenon (Belkin et al., 2019; Nakkiran et al., 2021) suggests that the goals of memorization and generalization are not contradictory and that a memorizing network can potentially also generalize well. A further investigation of this phenomenon in the context of conditional networks, and the design of conditional networks for practical datasets remains beyond the scope of the present work.

Notation: Unless specified otherwise, all vector variables are column vectors. We use ordinary font (such as u) for vectors, and the distinction between a vector and scalar will be clear from the context. The notation $f_n \in O(g_n)$ means that there is a constant $C > 0$ such that $f_n \leq C g_n$ for sufficiently large n. The set $\mathbb{R}^p$ is the set of all p-dimensional real-valued column vectors. The superscript $(\cdot)^T$ is the matrix transpose. The function $\mathbf{1}(\cdot)$ is the indicator function, and $\lceil \cdot \rceil$ is the ceiling operator. A function $f(x)$ of variable x is alternatively expressed as the mapping $x \mapsto f(x)$. Operator $\mathrm{rank}(\cdot)$ is the rank of a matrix. Finally, $\| \cdot \|$ is the Euclidean norm.

2 SYSTEM MODEL

2.1 UNCONDITIONAL FEEDFORWARD NETWORKS

Consider an ordinary unconditional feedforward network of a homogeneous set of neurons, all of which have the same functionality. For generality of the model, we allow skip connections. Formally, we consider an L-layer network with the input-output relationships

$$y_\ell = \phi\left(W_\ell \begin{bmatrix} y_{\ell-1} \\ \vdots \\ y_0 \end{bmatrix}\right), \ \ell = 1, \ldots, L, \tag{1}$$

where y_0 is the input to the neural network, y_ℓ is the output at Layer ℓ, and y_L is the neural network output. Also, W_ℓ is the weight matrix at Layer ℓ of appropriate dimensions. We consider the ReLU unit $\phi(x) = \max\{0, x\}$.

2.2 MEASURING THE COST OF COMPUTATION

A recurring theme of the paper will be to calculate the output of a neural network given an arbitrary input, however with low computational complexity. In particular, in (1), given the outputs of Layer $\ell - 1$, the calculation of the outputs of Layer ℓ can be accomplished through multiplication with matrix W_ℓ, followed by the activation functions. Each ReLU activation function can be implemented via a simple comparison given local fields. Hence, calculation of the output of Layer ℓ can be accomplished with $O(\mathrm{dim} W_\ell)$ operations (multiplications, additions, and comparisons), and the output y_L of the entire network can be calculated using $O(\sum_\ell \mathrm{dim} W_\ell)$ operations. Here, $\mathrm{dim} W_\ell$ represents the product of the number of rows and columns in W_ℓ. In other words, $\mathrm{dim} W_\ell$ is the number of dimensions in W_ℓ. Hence, in unconditional architectures, the number of operations to calculate the output of the network is essentially the same (up to constant multipliers) as the number of weights in the network. With regards to our basic measure of complexity, which relies on counting the number of operations, one can argue that multiplication is more complex than addition or comparison, and hence should be assigned a larger complexity. We hasten to note that the relative difficulty of different operations will not affect our final results, which will have an asymptotic nature.

It is instructive to combine the $O(\sum_\ell \mathrm{dim} W_\ell)$ complexity baseline with the memorization results described in Section 1. Let $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$ be a dataset of inputs. Let $d_1, \ldots, d_n \in \mathbb{R}^r$ be the corresponding desired outputs. In the memorization task, one wishes to design a network that can provide an output of d_i for an input of x_i for each $i \in \{1, \ldots, n\}$. It is known that, up to logarithmic factors, $O(\sqrt{n})$ weights and neurons are sufficient for memorization of n patterns (Vardi et al., 2022). It follows from the discussion in the paragraph above that $O(\sqrt{n})$ operations sufficient to recall a stored memory pattern (i.e. to calculate the output of the neural network for a given x_i). The goal of this paper is to accomplish the same task using much fewer operations required per input. Specifically, we will show how to do perfect recall using only $O(\log n)$ operations per input. We shall later show that $O(\log n)$ is, in fact, the best possible rate.

2.3 CONDITIONAL NETWORKS

In order to achieve perfect recall using sublinear number of operations, we use the idea of conditional computation. The conditioning model that we utilize in this work is a simple but general scenario where we allow executing different sub-neural networks depending on how an intermediate output of the network compares to some real number. Each conditioning is thus counted as one operation. An example conditional network is provided in Algorithm 1, where u represents the input, o is the output, and the subscripts are the vector components. For example, if the input is a vector u with $u_1 > 3$, resulting in an intermediate feature vector with $z_3 \neq 5$, then two operations are incurred due to the conditionings in Lines 1 and 5, and $O(\mathrm{dim}(W_1) + \mathrm{dim}(W_2) + \mathrm{dim}(W_5))$ operations are accumulated due to neural computations.

Algorithm 1 An example conditional neural network

1: **if** $u_1 > 3$ **then**
2: $\quad z = \phi(W_2 \phi(W_1 u))$
3: $\quad$ **if** $z_3 = 5$ **then**
4: $\quad\quad o = \phi(W_4 z)$
5: $\quad$ **else**
6: $\quad\quad o = \phi(W_5 z)$
7: $\quad$ **end if**
8: **else**
9: $\quad o = \phi(W_3 u)$
10: **end if**

3 SYNTHESIZING A CONDITIONAL NETWORK OUT OF AN UNCONDITIONAL NETWORK

3.1 THE SYNTHESIS THEOREM

Consider an arbitrary unconditional network as in (1), whose implementation requires $O(\sum_\ell \dim W_\ell)$ operations, as discussed in Section 2.2. Suppose that the network is well-trained in the sense that it can already provide the output d_i for a given input x_i, for every $i \in \{1, \ldots, n\}$. Out of such an unconditional network, we describe here a general methodology to synthesize a conditional network that requires much fewer than $O(\sum_\ell \dim W_\ell)$ operations, while keeping the input-output relationships $x_i \mapsto d_i$, $i \in \{1, \ldots, n\}$ intact. We need the following definition.

Definition 1. *Suppose that the activation function satisfies $\phi(0) = 0$. Given some fixed input to a neural network, a neuron with at least one non-zero input is called an active neuron. A neuron that is not active is called an inactive or a deactivated neuron. A weight is called active if it connects an active neuron to another active neuron. A weight is called inactive if it is not active.*

The source of the phrase "inactive" is the following observation: Consider an input to the neural network and the corresponding output. By definition, we will obtain the same output after removing all inactive neurons from the network for the same input. We can simply "ignore" inactive neurons. Likewise, we can remove any inactive weight and obtain the same output.

Our idea is to condition the computation on the set of active neurons and the corresponding active weights. Note that removing the inactive weights and neurons does not change the network output. Moreover, often the number of active weights given an input can be significantly lower than the overall number of neurons of the unconditional architecture (which determines the number of operations required to perform inference on the unconditional network), resulting in huge computational savings. The complication in this context is that the set of inactive weights depends on the network input. Fortunately, it turns out that determining the set of active weights can be accomplished with fewer operations than actually computing the local fields or outputs of the corresponding active neurons. The final result is provided by the following theorem.

Theorem 1. *Consider an arbitrary dataset $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$ of inputs. Let $d_1, \ldots, d_n \in \mathbb{R}^r$ be the corresponding desired outputs. Suppose that the unconditional neural network defined by (1) satisfies the desired input-output relationships in the sense that for any i, if the input to the network is x_i, then the output is d_i. Also suppose that the input x_i results in ω_i active weights. Then, there is a conditional network that similarly satisfies all desired input output relationships, and for every i, performs at most $C\omega_i$ operations given input x_i, where $C > 0$ is an absolute constant.*

Hence, a design criterion to come up with low computational-complexity neural networks might be to put a constraint on the number of active weights given any training input to an ordinary, unconditional feedforward architecture. By Theorem 1, we can synthesize a conditional neural network with the same functionality as the unconditional feedforward network. In the next section, we will apply this idea to the problem of memorization capacity.

We conclude this section by noting that Theorem 1 may prove to be useful in other applications. For example, any method that results in sparse representations or weights, such as pruning, will result in many inactive neurons and weights. Sparsity is useful; however, unstructured sparsity is difficult to exploit: For example, multiplying by a matrix half of whose entries are zero at random positions will likely be as difficult as multiplying by a random matrix without sparsity constraints. The construction in Theorem 1 may provide computational savings for such scenarios.

3.2 PROOF OF THEOREM 1

We describe the proof of Theorem 1 in two major steps as follows.

Step-1: By an " input configuration," we mean a certain subset of neurons that are active at a certain layer of the unconditional "base" network. We represent input configurations by binary vectors where "0" represents an inactive neuron, while "1" represents an active neuron. As an example, the input configuration $[1\,0\,1]$ refers to a scenario where only the first and the third neurons at the layer is active. We follow the convention that all components of the input vectors are "active" nodes.

The first step of the proof is to construct what we call a "configuration tree." The configuration tree is a directed rooted tree where the nodes at depth ℓ correspond to all input configurations induced by the n possible inputs at Layer ℓ, where $\ell \in \{1, \ldots, L\}$. The only exception is that there is only one input configuration at depth 0, corresponding to the input layer of the neural network, which is represented by an all-1 vector. This unique input configuration becomes the root node of the tree.

Analogous to input configurations, "output configurations" are binary vectors that represent whether neurons provide zero or non-zero outputs. For example, an output configuration of $[1\,1\,0]$ means that only the first and the second neurons provide a non-zero output.

We now construct the configuration tree by showing the inputs to the neural network one by one. Let $\alpha_{i\ell}$ denote the number of active nodes at layer ℓ when input x_i is shown to the network. Each active node at Layer 1 will assume either a zero or non-zero output, resulting in $2^{\alpha_{11}}$ edges pointed away from the root node. Each of the $2^{\alpha_{11}}$ edges originating from the root node represents the $2^{\alpha_{11}}$ possible output configurations at Layer 1. The endpoint of each edge represents an implied input configuration at Layer 2. Note that, although a particular input like x_1 will induce only one output configuration, we include all possible output configurations for a given input configuration when constructing the configuration tree.

We then find the tree node at Depth 2 corresponding to the input configuration for x_1 at Layer 2. We add $2^{\alpha_{12}}$ edges pointed away from this node, representing all possible output configurations at Layer 2. We continue in the same manner for all subsequent layers by adding $2^{a_{1\ell}}$ branches to Depth ℓ, while ℓ increases up to L. The tree is grown for the subsequent samples in the same manner.

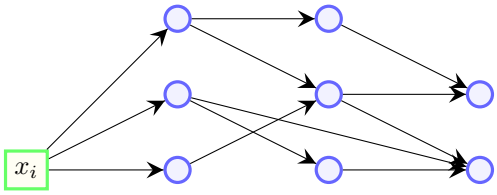


Figure 1: An example network for demonstrating the construction of the configuration tree.

Example 1. As an example of the construction of the configuration tree, consider the network in Fig. 1 with one dimensional inputs and two dimensional outputs. We consider a dataset with two samples $n = 2$. Suppose that the first input $x_1 \neq 0$ provides the input configurations $[1\,1\,1]$, $[1\,1\,0]$, $[1\,0]$, and output configurations $[1\,0\,1]$, $[1\,0\,0]$, $[1\,0]$, at Layers 1, 2, 3, respectively. Then, after processing x_1, the tree will consist of all blue/white nodes in Fig. 2 and the edges between them. Note that the vertices represent the input configurations, while edges represent the output configurations. Hence, the node `110` at Depth 1 of the tree represents the scenario where only the first and the second neurons in the first layer are active (have at least one non-zero input). Since there are 2 active neurons, there are 4 possible output configurations, represented by the 4 edges pointed away from the node. In

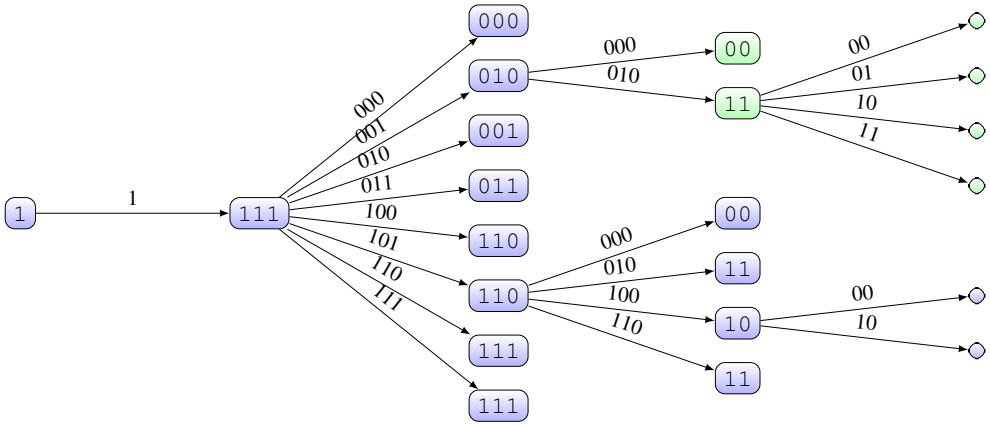


Figure 2: One configuration tree corresponding to the network in Fig. 1.

particular, the output configuration 010 implies an input configuration of 11 at Depth 2 (Layer 3). This is because, Node 2 of Layer 2 is connected to both neurons at Layer 3, and a non-zero output of this neuron implies a non-zero input to Layer 3 neurons. Now, suppose the second input $x_2 \neq 0$ provides the input configurations $[1\,1\,1]$, $[0\,1\,0]$, $[1\,1]$, and output configurations $[0\,0\,1]$, $[0\,1\,0]$, $[1\,1]$, at Layers 1, 2, 3, respectively. Then, the construction of the tree is completed as shown in Fig. 2.□

Step-2: Consider now the conditional network constructed via the configurationtree in the following manner. First, we check the 2^d possibilities of the output configurations of the input layer via d nested conditioning statements. An example of this step is provided in Algorithm 2 for a two-dimensional input $[u_1 u_2]^T$. After each of the d conditionings, one can obtain a unique input configuration at Layer 1 of the (unconditional) network. In other words, we can uniquely determine which nodes will become active at Layer 1. Only the local fields of the active neurons have to be calculated, resulting in $O(d\alpha_{i1})$ operations for input with index i. Once the local fields are calculated, we can check the output configurations of the first layer via nested conditioning statements in the same manner as it was done for the input layer. For calculation of the local fields at Layer 2 and above, we only consider inputs from active neurons, so that we only utilize active weights eventually. Continuing in this manner for subsequent layers, the configuration tree provides a blueprint for constructing the conditional network out of the unconditional network. By construction, the total number of operations for calculating the output for a given input x_i is bounded above by a constant multiple of ω_i.

Algorithm 2 Two nested conditionings

```
if u_1 = 0 then
    if u_2 = 0 then
        ...
    else
        ...
    end if
else
    if u_2 = 0 then
        ...
    else
        ...
    end if
end if
```

4 Memorization with Low Computational Complexity

In this section, we introduce our ReLU network to memorize the input-output relationships $x_i \mapsto d_i$, $i = 1, \ldots, n$. Our network will only yield $O(\log n)$ active neurons and weights per input. Application of Theorem 1 will then imply a conditional network that can achieve perfect recall with only $O(\log n)$ operations per input, as desired.

Given a neural network f, let $A(x; f)$ denote the number of active neurons for an input x to f. Our feedforward ReLU network construction is provided by the following theorem.

Theorem 2. *Let $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$ be a dataset of input patterns such that for every $i, j \in \{1, \ldots, n\}$ with $i \neq j$, we have $x_i \neq x_j$. Let $\bar{x}_i = \left[\begin{smallmatrix} 1 \\ x_i \end{smallmatrix}\right] \in \mathbb{R}^q$, where $q = 1 + p$, be the augmented dataset patterns for biasing purposes. Let $d_1, \ldots, d_n \in \mathbb{R}^r$ be the corresponding desired outputs. Suppose every component of d_i is non-negative for each $i \in \{1, \ldots, n\}$. Then, there is a neural network $f : \mathbb{R}^q \to \mathbb{R}^r$ such that for every $i \in \{1, \ldots, n\}$, we have $f(\bar{x}_i) = d_i$, and*

$$\begin{aligned} A(\bar{x}_i; f) &\leq (q + r + 2)\lceil \log_2 n \rceil + q + r - 1 \\ &\in O((q + r) \log n). \end{aligned} \tag{2}$$

Moreover, every neuron in network f has at most q weights, resulting in $O(q(q + r) \log n)$ active weights per input according to (2).

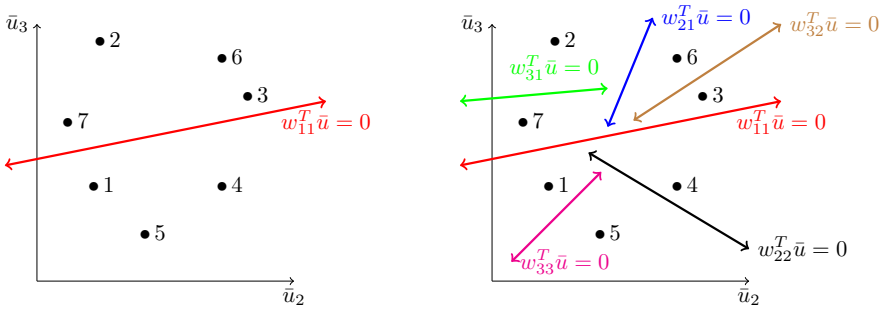
Proof. (Sketch) We provide an illustrative sketch of the proof. Suppose we wish to memorize the two-dimensional dataset given by Fig. 3a. We can divide the datasets to two parts via a line, the resulting two parts to further two sub-parts, and so on, until reaching singletons, as shown in Fig. 3b. The overall network architecture that achieves the performance in the statement of the theorem is then shown in Fig. 4. The initial block $\mathbf{T}$ is a basic preprocessing translation. The "switch" $\mathbf{S}_{ij}$ corresponds to the line parameterized by weights w_{ij} in Fig. 3b. The switch $\mathbf{S}_{ij}$ routes the zero vector to one output path and copy its input to the other output path, depending on the side of the w_{ij}-line its input vector resides. The switches are followed by ReLU neurons with weights γ_i. These neurons map the corresponding input pattern on the active path to its desired output. Finally, the output of the γ_i-neurons are accumulated. As an example, the signals on the graph for input pattern $\bar{x}_6$ is provided, with $\bar{\bar{x}}_6 \triangleq \mathbf{T}(\bar{x}_6)$. In general, all input-output relationships are satisfied with $O(\log n)$ active neurons per dataset sample. We refer to the complete proof in Appendix A for the precise bounds. □

Corollary 1. *Let x_i, d_i, $i = 1, \ldots, n$ be a sequence of arbitrary input-output pairs as stated in Theorem 2. Then, there is a conditional network that, for every i, provides an output of d_i whenever the input is x_i by performing only $O(\log n)$ operations.*

Proof. We apply the synthesis procedure in Theorem 1 to the network constructed in Theorem 2. □

We recall that for ReLU networks without conditional computation, the best achievability result (Vardi et al., 2022) requires $O(\sqrt{n \log n})$ neurons and weights for memorizing a dataset of size n. Since the construction in Vardi et al. (2022) is not optimized for conditional computation, it is easily observed that every input activates all $O(\sqrt{n \log n})$ neurons and weights of the network, resulting in $O(\sqrt{n \log n})$ active weights per input. In fact, the remarkable construction in Vardi et al. (2022) is a narrow network of a finite width of only 12, but depth $O(\sqrt{n \log n})$. Even if every input activates only one neuron at each layer, one obtains $O(\sqrt{n \log n})$ active neurons or arithmetic operations per input. In contrast, we only need $O(\log n)$ active weights or operations (via Corollary 1) per input.

The fact that one cannot fundamentally achieve a sub-polynomial number of neurons without conditional computation is an easy consequence of the VC dimension theory. In our setting, the VC dimension corresponds to the cardinality of the largest dataset that can be memorized. Hence, upper bounds on the VC dimension translate to lower bounds on the memorization capacity. In this context, Anthony et al. (1999, Theorem 8.6) provides an $O(n_w^2)$ bound on the VC dimension for ReLU networks, where n_w denotes the number of weights in the network. It follows that $O(\sqrt{n})$ weights are the best possible for ReLU networks without conditional computation, meaning that the results of Vardi et al. (2022) are optimal up to logarithmic factors. Also, using the bound $n_w \leq n_e^2$, where n_e is the number of weights of the network (equality holds in the extreme scenario where every neuron is connected to every other neuron, including itself), we obtain the necessity of $O(n^{1/4})$ weights.



(a) Dividing a set of point to two equal subsets. (b) Continued divisions until reaching singletons.

Figure 3: The divide and conquer strategy.

In contrast to the deep and narrow architecture that is optimal for unconditional computation, the network that achieves the performance in Theorem 2 is considerably wider but much shallower. In fact, the proof in Appendix A reveals that our network has width $O(n)$, with $O(n)$ nodes, and depth $O(\log n)$. These numbers are a consequence of the classical binary decision tree that we utilize in our network construction. Thanks to conditional computation, although the network has $O(n)$ nodes, every dataset pattern activates only $O(\log n)$ neurons instead of $O(\sqrt{n})$ neurons without conditional computation. Note that the function $\log n$ grows much slower than $\sqrt{n}$ so that the number of neurons activated by the conditional computation is asymptotically negligible relative to an unconditional network. Moreover, every neuron in the conditional computation network has a bounded number of q weights. The two facts translate to big savings in terms of the cost of computation, thanks to Theorem 1. An interesting open problem that we shall leave as future work is whether one can achieve the same performance with a much smaller network size, e.g. with $O(\sqrt{n})$ neurons, which is known to be optimal. This will also help reduce the size of the network synthesized by Theorem 1.

It should be mentioned that the bound (2) on the number of active neurons as stated in Theorem 2 holds only for input patterns that belong to the dataset. For such patterns, only one path from the input to the output of the neural network is activated. The global behavior of the neural network for arbitrary inputs is more complex. A careful analysis of the construction in Appendix A will

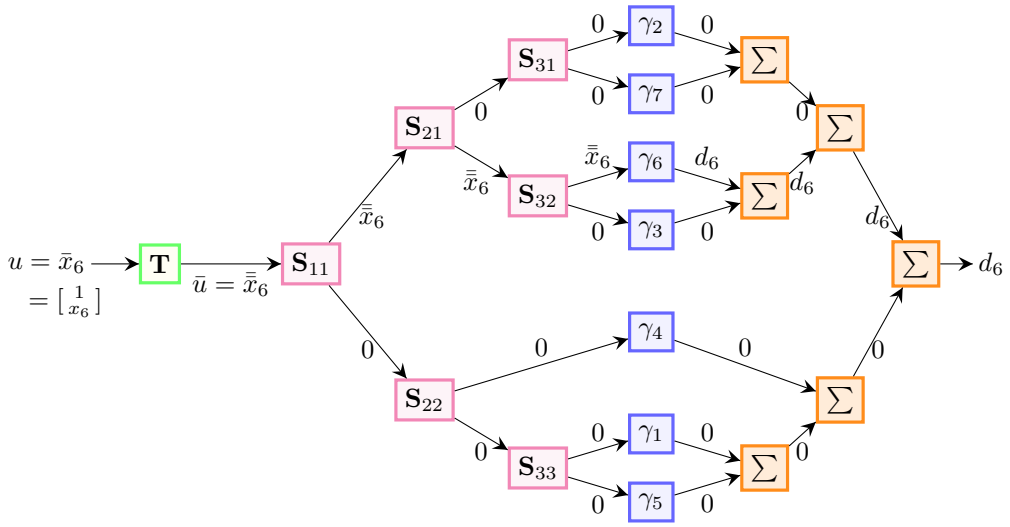


Figure 4: An example network architecture for the achievability result. The block $\mathbf{T}$ represents the transformation in Step 1. Blocks $\mathbf{S}_{ij}$ are the routing switches. Blocks γ_i represent ReLU neurons with weights γ_i, and $\sum$ blocks represent ReLU neurons with all-one weights.

reveal that near the decision boundaries (the lines in Fig. 3b), multiple paths of the network will be activated, resulting in more active neurons than the upper bound in (2). However, the measure of such pathological inputs can be made arbitrarily small by tuning the switches appropriately.

5 Ultimate Computational Limits to Memory Recall

In the previous section, we showed that $O(\log n)$ operations is sufficient to perfectly recall one of n input-output relationships. We now analyze the necessary number of operations per input for successful recall. Our main result in this context is provided by the following theorem.

Theorem 3. *Let the input vectors $x_1, \ldots, x_n \in \mathbb{R}^p$ and the corresponding desired output vectors $d_1, \ldots, d_n \in \mathbb{R}$ satisfy the following property:*

- *The matrix $\begin{bmatrix} x_{i_1} & \cdots & x_{i_{p+1}} \\ d_{i_1} & \cdots & d_{i_{p+1}} \end{bmatrix}$ has rank $p+1$ for any subset $\{i_1, \ldots, i_{p+1}\}$ of $\{1, \ldots, n\}$.*

Suppose that a conditional network f satisfies the desired input-output relationships in the sense that for every i, the output of the network is d_i whenever the input is x_i. Also, assume that the number of operations performed on each x_i is bounded above by some $\alpha \geq 0$. Then, we have $\alpha \geq \log_2 \frac{n}{p} \in O(\log n)$.

Proof. Since there are at most α operations per input, there are at most α comparisons per input as well, counting the comparisons needed to implement the neuron activation functions. We can represent the entire conditional computation graph/network as a binary tree where the distance between the root and a leaf node is at most α. This results in tree of at most 2^α leaf nodes. Each node of the tree compares real numbers to intermediate variables, which are linear functions of network inputs or other intermediate variables. Assume now the contrary to the statement of the theorem, i.e. the number of operations for every input is at most α, all input-output relationships are satisfied, but $n > 2^\alpha p$. Since there are at most 2^α leaf nodes, there should be at least one leaf node that admits $1+p$ inputs (i.e. there are $1+p$ input patterns of the dataset such that the corresponding path over the tree ends at the leaf node). Without loss of generality, suppose the indices for these inputs are the first $1+p$ natural numbers. Writing down the input output relationship of the network for the leaf node, we obtain

$$[d_1 \cdots d_{1+p}] = W_0 [x_1 \cdots x_{1+p}] \tag{3}$$

for some $q \times p$ matrix W_0. This relationship follows, as by fixing a path on the tree, we obtain the unique linear transformation W_0 that maps the inputs to the neural network to its outputs. According to the Rouché–Capelli theorem (Shafarevich & Remizov, 2012, Theorem 2.38), a necessary condition for the existence of W_0 to solve (3) is

$$\operatorname{rank}([x_1 \cdots x_{p+1}]) = \operatorname{rank}\left(\begin{bmatrix} x_1 & \cdots & x_{p+1} \\ d_1 & \cdots & d_{p+1} \end{bmatrix}\right). \tag{4}$$

On the other hand, as a result of the rank condition stated in the theorem, the left hand side rank evaluates to p, while the right hand side evaluates to $1 + p$. We arrive at a contradiction, which concludes the proof of the theorem. □

The condition on the dataset and the desired outputs that appear in the statement of Theorem 3 is, up to a certain extent, necessary for the lower bound to hold. For example, if the desired outputs can simply be obtained as a linear transformation of inputs, then one only needs to perform a constant number of operations for each input to obtain the desired outputs, and the lower bound becomes invalid. In this context, the rank condition ensures that subsets of outputs cannot be obtained as linear functions of inputs. However, it should also be mentioned that the condition is not particularly restrictive in limiting the class of datasets where the theorem holds. For example, if the components of the dataset members and the corresponding desired outputs are sampled in an independent and identically distributed manner over any continuous distribution with positive support, it can be shown that the rank condition will hold with probability 1. Hence, it can be argued that almost all datasets satisfy the rank condition and thus obey the converse result.

Corollary 1 has shown that n patterns can be memorized using a conditional network that performs $O(\log n)$ operations per input. The matching $O(\log n)$ lower bound in Theorem 3 proves that the $O(\log n)$ rate is the best possible. However, the two results do not resolve how the number of active neurons should scale with respect to the input and output dimensions.[1] This aspect of the problem is left as future work.

6 Conclusions

We have studied the fundamental limits to the memorization capacity of neural networks with conditional computation. First, we have described a general procedure to synthesize a conditional network out of an ordinary unconditional feedforward network. According to the procedure, the number of operations required to perform inference on an input in the synthesized conditional network becomes proportional to the number of so-called "active weights" of the unconditional network given the same input. This reduces the problem of designing good conditional networks to the problem of designing ordinary feedforward networks with a low number of active weights or nodes. Using this idea, we have shown that for ReLU networks, $O(\log n)$ operations per input is necessary and sufficient for memorizing a dataset of n patterns. An unconditional network requires at least $O(\sqrt{n})$ operations to achieve the same performance.

One direction for future work is to study the memorization capacity for a sum-constraint, as opposed to a per-input constraint on the number of operations. While a per-input constraint makes sense for delay-sensitive applications, the sum-constrained scenario is also relevant for early-exit architectures, where there is a lot of variation on the size of active components of the network. Extensions of our results to different activation functions or to networks with bounded bit complexity can also be considered. Also, the main goal of this paper has been to derive theoretical bounds on the memorization capacity. More research is clearly needed for practical methods to train neural networks that can effectively utilize conditional computation and also generalize well. We hope that the constructions and theoretical bounds provided in this paper will motivate further research in the area.

[1] Although Theorem 3 only considers scalar desired outputs, it can easily be extended to the multi-dimensional case. In fact, a successful memorization of, say, two-dimensional output vectors with $o(\log n)$ active neurons would imply the successful memorization of scalar outputs with the same number of neurons (simply by ignoring the neurons that provide the second component of the output), contradicting Theorem 3.

REFERENCES

Martin Anthony, Peter L Bartlett, Peter L Bartlett, et al. *Neural network learning: Theoretical foundations*. Cambridge University Press, 1999.

Eric B Baum. On the capabilities of multilayer perceptrons. *Journal of Complexity*, 4(3):193–215, 1988.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Kyunghyun Cho and Yoshua Bengio. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362*, 2014.

Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*, 2013.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations (ICLR)*, May 2019.

Alperen Gormez and Erdem Koyuncu. Class means as an early exit decision mechanism. *arXiv preprint arXiv:2103.01148*, 2021.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. In *International Conference on Learning Representations*, 2016.

Guang-Bin Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281, 2003.

Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *Microsoft Research Technical Report*, 2016. URL https://arxiv.org/pdf/1603.01250.pdf.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning*, June 2019.

Minkyu Kim and Jae-Sun Seo. An energy-efficient deep convolutional neural network accelerator featuring conditional computing and low external memory access. *IEEE Journal of Solid-State Circuits*, 56(3):803–813, 2020.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *IEEE International Conference on Computer Vision*, December 2015.

Oded Z Maimon and Lior Rokach. *Data mining with decision trees: theory and applications*. World scientific, 2014.

GJ Mitchison and RM Durbin. Bounds on the learning capacity of some multi-layer networks. *Biological Cybernetics*, 60(5):345–365, 1989.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

Sejun Park, Jaeho Lee, Chulhee Yun, and Jinwoo Shin. Provable memorization via deep neural networks using sub-linear parameters. In *Conference on Learning Theory*, August 2021.

Shashank Rajput, Kartik Sreenivasan, Dimitris Papailiopoulos, and Amin Karbasi. An exponential improvement on the memorization capacity of deep threshold networks. *Advances in Neural Information Processing Systems*, December 2021.

Igor R Shafarevich and Alexey O Remizov. *Linear algebra and geometry*. Springer Science & Business Media, 2012.

Eduardo Sontag. Remarks on interpolation and recognition using neural nets. *Advances in Neural Information Processing Systems*, 3, 1990.

Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, June 2019.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition*, December 2016.

Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*. 2015.

Gal Vardi, Gilad Yehudai, and Ohad Shamir. On the optimal memorization power of ReLU neural networks. In *International Conference on Learning Representations*, April 2022.

Roman Vershynin. Memory capacity of neural networks with threshold and rectified linear unit activations. *SIAM Journal on Mathematics of Data Science*, 2(4):1004–1033, 2020.

Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. In *International Conference on Learning Representations*, April 2017.